Network Working Group
Request for Comments: 798

A. Katz
ISI
September 1981

DECODING FACSIMILE DATA FROM THE RAPICOM 450

I. Introduction

This note describes the implementation of a program to decode facsimile data from the Rapicom 450 facsimile (fax) machine into an ordinary bitmap. This bitmap can then be displayed on other devices or edited and then encoded back into the Rapicom 450 format. In order to do this, it was necessary to understand the how the encoding/decoding process works within the fax machine and to duplicate that process in a program. This algorithm is descibed in an article by Weber [1] as well as in a memo by Mills [2], however, more information than is presented in these papers is necessary to successfully decode the data.

The program was written in L10 as a subsystem of NLS running on TOPS20. The fax machine is interfaced to TOPS20 as a terminal through a microprocessor-based interface called FAXIE.

Grateful acknowledgment is made to Steve Treadwell of University College, London and Jon Postel of Information Sciences Institute for their assistance.

II. Interface to TOPS20

The fax machine is connected to a microprocessor-based unit called FAXIE, designed and built by Steve Casner and Bob Parker. More detailed information can be found in reference [3]. FAXIE is connected to TOPS20 over a terminal line, and a program was written to read data over this line and store it in a file. The decoding program reads the fax data from this file.

The data comes from the fax machine serially. FAXIE reads this data into an 8-bit shift register and sends the 8-bit byte (octet) over the terminal line. Since the fax machine assigns MARK to logical 0's and SPACE to logical 1's (which is backward from RS232), FAXIE complements each bit in the octet. The data is sent to TOPS20 in octets, the most significant bit first. If you read each octet from most significant bit to least significant bit in the order FAXIE sends the data to TOPS20, you would be reading the data in the same order in comes into FAXIE from the fax machine.

The standard for storing Rapicom 450 Facsimile Data is described in RFC 769 [4]. According to this standard, each octet coming from FAXIE must be complemented and inverted (i.e. invert the order of the bits in the octet). Thus, the receiving program did this before

storing the data in a file. When the decoding program reads this file, it must invert and complement each octet before reading the data.

Each data block from the fax machine is 585 bits long. The end of this data is padded with 7 0's to make 592 bits or 74 octets. According to RFC 769, this data is stored in a file preceded by a length octet and a command octet. The possible commands are:

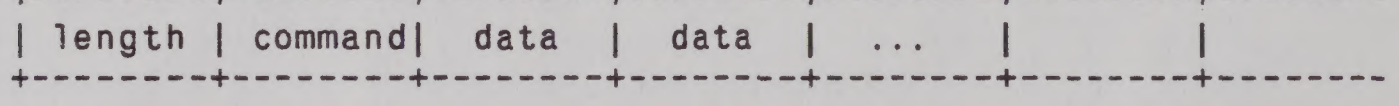
56 (70 octal)--This is a Set-Up block (the first block of the file, contains information about the fax image)

57 (71 octal)--This is a data block (the rest of the blocks in the file except for the last one)

58 (72 octal)--End command (the last block of the file)

The length field tells how many octets in this block and is always 76 (114 octal) except for the END command which can be 2 (no data). The length and command octets are NOT inverted and complemented.

Below is a diagram of each block in the file:



III. The Rapicom 450 Encoding Algorithm

An ordinary 8 1/2" by 11" document is made up of about 2100 scan lines, each line has 1726 pels (picture elements) in it. Each pel can be either black (1) or white (0).

The Rapicom 450 has three picture quality modes. In fine detail mode, all of the document is encoded. In quality mode only every other scan line is encoded and it is intended that these missing lines are filled in on playback by replicating the previous line. There is also express mode, where only every third line is encoded.

Data is encoded two lines at a time, using a special two dimensional run-length encoding scheme. There are 1726 pels on top and 1726 pels on the bottom. Each pair (top-bottom) of pels is called a column. For each of the 1726 columns you can have any one of four configurations (called states):

column (top-bottom)	pels	state
W-W	0,0	0
W-B	0,1	1
B-W	1,0	2
B-B	1,1	3

The encoding algorithm can be described in terms of a non-deterministic finite-state automaton shown in Fig. 1 (after Mills [2]). You start out in a state (0-3) and transform to another state by emitting the appropriate bits marked along the arcs of the diagram. For example, suppose you are in state 1 (WB). To go to state 2 (BW), you would output the bits 101 (binary); to go to state 0 (WW) you would output the bits 1000. Note that the number of bits on each transition is variable.

In states 0 (WW) and 3 (BB), a special run length encoding scheme is used. There are two state variables associated with each of these states. One variable is a run-length counter and the other is the field length (in bits) of this counter. Upon entry to either of these two states, the counter is initialized to zero and is incremented for every additional column of the same state. At the end of the run, this counter is transmitted, extending with high order zeros if necessary. If the count fills up the field, it is transmitted, the field length is incremented by one, and the count starts again. This count is called the run length word and it is between 2 and 7 bits long.

For example, suppose we are in state 0 (WW) and the run length for this state (refered to as the white run length) is 3. Suppose there are three 0's in a row. The first 0 was encoded when we came to this state, there are two more 0's that must be encoded. Thus we would send a 010 (binary). Similarly, if there are seven 0's in a row, we would send a 110, but eight 0's would be sent by 111 followed by 0000 and the white run length becomes 4. (Ten 0's would be encoded as 111 followed by 0010 and the white run length would be 4).

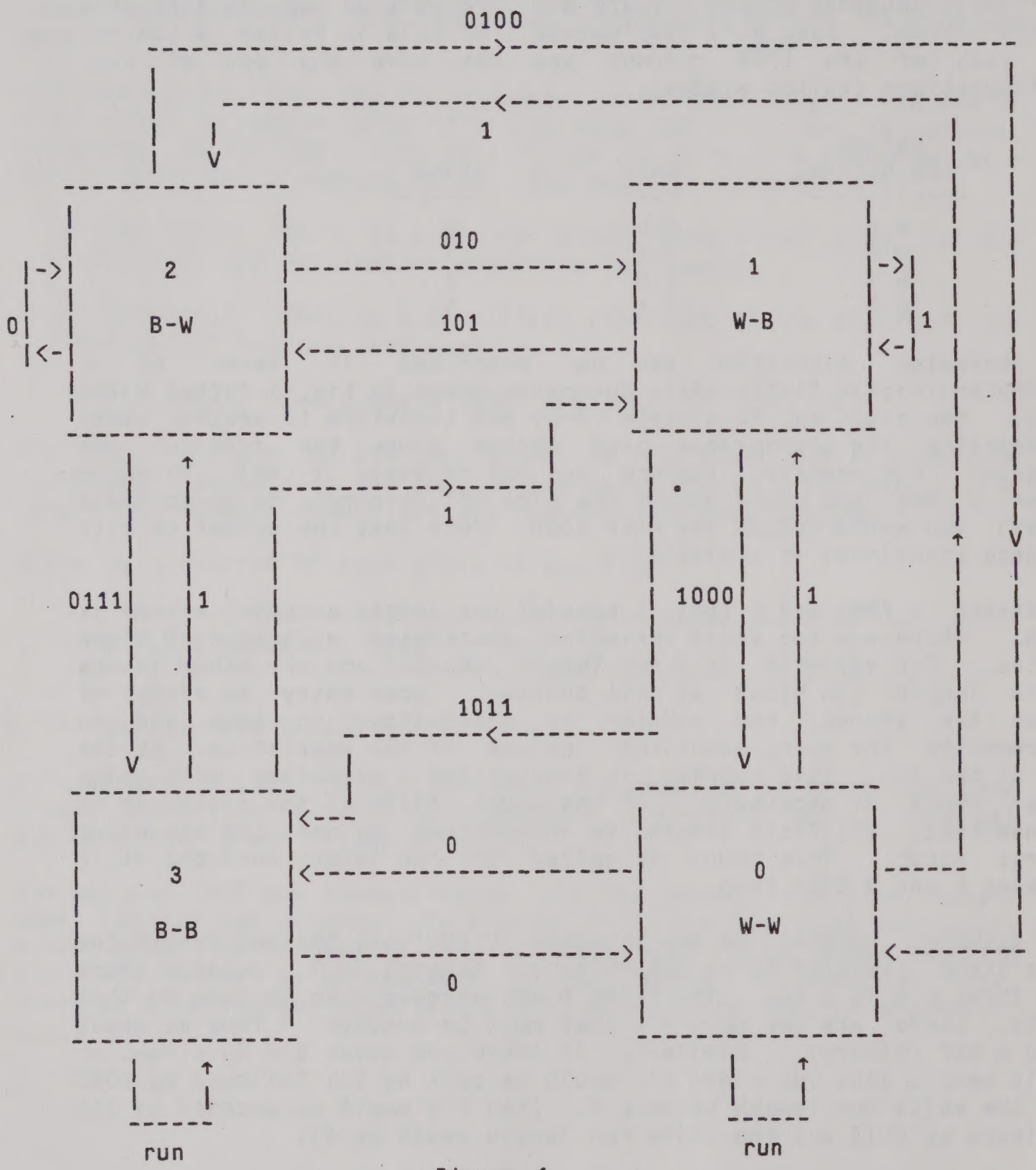


Figure 1.
Non-deterministic finite-state machine diagram for RAPICOM 450

Run length word lengths must be between 2 and 7. The field length is decremented if the run is encoded in one word and:

1. If the run length is 3 and the highest order bit is 0.

2. Or, if the run length is 4, 5, 6, or 7 and the highest order 2 bits are 0.

In addition to all this, there is a special rule to follow if the run occupies at least two run words (and can involve incrementing the run word size) and the run ends exactly at the end of a scan line. In this case, the last word of the run is tested for decrement as if the previous words in the run did not exist.

An Example:

To confirm the reader's understanding of the encoding procedure, suppose we had the following portion of a document (1=black, 0=white):

```
top row:      0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 ...
bottom row:   1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 ...
-----------   --------------------------------
state:        1 3 3 3 3 2 0 0 0 0 0 2 2 1 0 0 ...
```

Suppose also that the black run field length is 2, the white run length is 3, and the state is 1. (This example comes from reference [1].)

This portion would be encoded as:

```
1 1011 11 000 1 0100 100 1 0 010 1000 ...
```

NOTE: It turns out that the Rapicom 450 sends the bits of a field in reverse order. This will be discussed in the section V. However, since each run length field is sent reversed, the above encoded bit pattern would actually be sent as:

```
1 1011 11 000 1 0100 001 1 0 010 1000 ...
                     ↑
                     |-this is actually 100 reversed
```

Another Example:

This example illustrates the rule for decrementing the run length word lengths:

```
top row:      0 1 1 0 0 1 1 1 1 1 0 0 ...
bottom row:   1 1 1 1 1 0 1 1 1 1 1 0 ...
-----------   ----------------------
state:        1 3 3 1 1 2 3 3 3 3 1 0 ...
```

Here, let us suppose that the black run field length is now 4, the white is still 3, and the state is 1.

This portion would be encoded as:

```
1 1011 0001 1 1 101 0111 011 1 1000 ...
       ↑                 ↑
       |-goes to 3       |-blk cnt goes to 2
```

When we reverse the order of the run fields, the bit pattern that is actually sent is:

```
1 1011 1000 1 1 101 0111 110 1 1000 ...
       ↑
       |-this is actually 0001 reversed, etc.
```

IV. The Setup Block and the Data Header

Each data block from the fax machine is 585 bits long. The number of blocks in a picture is variable and depends on the size and characteristics of the picture. It should be emphasized that a block can end in the middle of a scan line of the document. There can in fact be many scan lines in a block.

The 585 bit data block is composed of a 24 bit sync code which is used to recognize the beginning of a block, a 37 bit header, 512 bits of actual data, and a 12 bit CRC checksum:

24-bit sync code	37-bit header	512-bit data	12-bit checksum

The number of useful data bits is variable and can be between 0 and 512 (although there are always 512 bits there, some of them are to be ignored). The number of data bits to be used is given in the header.

The 37 bits of header is composed of:

2-bit seq num	5-bit flags	10-bit data count	12-bit x position	3-bit black size	3-bit white size	2-bit state

An explanation of these fields follows:

IMPORTANT NOTE: Most (but not all) of these fields are sent by the fax machine in REVERSE ORDER. The order of each n-bit field must be inverted.

Sync code

This is used to synchronize on each block. The value of this 24 bit field is 30474730 octal (not reversed).

Sequence number

This number cycles through 0, 1, 2, 3 for the data blocks. It is 0 for the Set-Up block (not reversed).

Flags

Each of these flags are 1 bit wide:

Run

Purpose unknown, it always seems to be 1.

Cofb

Purpose unknown, it always seems to be 0.

Rpt

1 for Set-Up blocks (which are repeated when coming from the fax machine though only one of them is transfered by FAXIE to TOPS20 and stored in the file) and 0 for data blocks.

Spare

Purpose unknown, doesn't matter.

Sub

1 if this is a Set-Up block.

Data Count

Number of useful bits to use out of the 512 data bits. NOT ALL of the 512 data bits are used, only this number of them. This number can be 0 (usually in one of the first data blocks) which means to throw away this block. (This field is reversed!)

X Position

Current position on the scan line, a value between 0 and 1725. If this number is greater than where the previous block left off, the intervening space should be filled with white (0's). If this number is less than where the previous block left off, set the X position to this value and replace the overlapped data with the new data from this block. If this number is greater than 1726, ignore this field and continue from where the previous block left off. (This field is reversed!)

Black Size

The size of the black run length field, must be between 2 and 7. This is the correct value for the black size. It may differ from what was found at the end of the previous block. (This field is reversed!)

White Size

The size of the white run length field, must be between 2 and 7. It may differ from what was found at the end of the previous block. (This field is reversed!)

State

The current state. This is the correct state. It may differ from the state at the end of the previous block. (This field is not reversed.)

Data

512 bits of the actual encoding of the document. NOT ALL of this data is used in general, only the amount specified by the

data count. If this is a set up block, the data contains information about the type of document (see below).

Checksum

CRC checksum on the entire block. Uses polynomial x**12+x**8+x**7+x**5+x**3+1.

In a setup block, the data portion of the data block consists of:

6-bit flags	5-bit spare	1-bit multi page	20-bits of zeros	480-bits 1's and 0's

Specifically these are:

6 flags (each are 1 bit):

Start bit

Always 0.

Speed

Is 1 if express mode.

Detail

Is 1 if detail mode. (NOTE: If the Detail and Speed flags are both 0, then data is in Quality mode).

14 inch paper

is 1 if 14 inch paper length.

5.5 inch paper

is 1 if 5.5 inch paper length. (NOTE: If the 14 inch and 5 inch flags are both 0, then paper length is 11 inch).

paper present

is 1 if paper is present at scanner (should be always 1).

Spare:

These 5 bits can be any value.

Multi-page:

1 if multi page mode

Rest of data of set-up block:

The above fields are followed by twenty 0 bits and the rest of the 512 bits of the block are alternating 0's and 1's.

There are a number of important points to be remembered in regard to the header of a data block. First of all, the data count, the x-position, and the black and white run sizes must be read IN REVERSE ORDER. The reason for this is that the fax machine sends these bits in reverse order. However, the sequence number and the state fields ARE NOT REVERSED. In addition to this, each run field in the data IS REVERSED. This reversing of the bits in each n-bit field is completely separate from the reversing and complementing of each octet mentioned earlier.

Second, only the first n bits, where n is the value of the data count field (remember its reversed!), of the data is valid, the rest is to be ignored. If n is zero, the whole block is to be ignored.

Third, if the x position is beyond where the last block ended, fill the space between where the last block ended and the current x postion with white (0's). If the x postition is less then where the last block ended, replace the overlapped data with the data in the new block. If the x postition is greater than 1726, ignore it and continue from where the previous block left off.

Fourth, the black size, white size (reversed), and state (not reversed!) given in the header are the correct values even if they disagree with the end of the previous block.

Finally, the sequence number (not reversed) should count through 0,1,2,3. If it does not, a block is missing.

V. The Decoding Algorithm

Upon first glance at the finite state diagram in Figure 1, it may seem that it would be difficult to create a decoding procedure. For example, if you are in the WW state, and the next bit is a 1, how do you know whether to do a transition to WB or BW? The answer to this is to recognize that every arc out of the BW state begins with 0 and every arc out of WB begins with 1. Thus, if you are in the WW state, and the next bit is 1, followed by a 0, you know to go to the BW state. If the next bit is 1, followed by a 1, you know to go to the WB state.

In writing the decoding program it was necessary to have two ways of reading the next bit in the data stream. The first way reads the bit and "consumes" it, i.e. increments the bit pointer to point at the next bit. The other way does not "consume" it. Below are four statements which show how to decode fax data. The numbers in parentheses are not to be consumed, that is to say they will be read again in making the next transition.

If I am in state BW (2) and the next bits are:

0 (0):	go to BW
0111:	go to BB
010 (1):	go to WB
0100:	go to WW

If I am in state WB (1) and the next bits are:

1 (1):	go to WB
1000:	go to WW
101 (0):	go to BW
1011:	go to BB

If I am in state WW (0), then first go through the run length algorithm, then if the next bits are:

0:	go to BB
1 (0):	go to BW
1 (1):	go to WB

If I am in state BB (3), then first go through the run length algorithm, then if the next bits are:

0:	go to WW
1 (0):	go to BW
1 (1):	go to WB

For the run length algorithm, remember, look at the next n bits, where n is the length of either the black or white run length

word, REVERSE the bits, and output that many BB's or WW's (depending on whether black or white run). If the field is full, increment the size of the word, and get that many bits more, i.e. get the next n+1 bits, etc. Also, the run length word length can be decremented according to the rules given in section III.

You always go through the run length even if there is only one WW or BB, in this case, the run field will be 0.

Let us look at the first example given in section III. Suppose we want to decode the bits: 1101111000101001001001010000... (we have already reversed the run lengths to make things easier).

We are in state 1 (WB) and the black run length word length is 2 and the white length is 3. We get these initial values either from the block header, or by remembering them from the previous transitions if this is not the start of the block. According to our rules, we would parse this string as follows:

1(1) 1011 11 000 1(0) 0100 100 1(0) 0(0) 010(1) 1000...

The numbers in parentheses are numbers that were read but not "consumed", thus the next number in the sequence is the same as the one in parentheses. First, we see a 1 and that the next bit is a 1, this means that we go to WB. Then we have a 1011 which means to go to BB. Then we do a run, we have a 11 followed by a 000 which means the black run length gets incremented by 1 (it is now 3) and we get 3 MORE BB's. Now we have a 1 followed by 0 which means go to BW. Next a 0100 which is WW. Then we have a run, 100, which means four more WW's. We keep going like this and we get the original bit pattern given in the first example of section III.

It is important to always start fresh when dealing with each block. There are many reasons for this. The first is that sometimes blocks are dropped, and you can recover from this if you resynchronize at the start of each block. Also, if at the end of the previous block, there is about to be a transition, instead of making it at the beginning of the next block, the fax machine gives the new state in the header of the next block and goes from there. Thus it is important to always start at whatever state is given in the header, and to align yourself at the current X position given there also.

Sometimes, while decoding a block, a bit pattern will occur which

does not correspond to any transition. If this happens, the rest of the block may be bad and should be discarded.

The decoding program decodes the fax data block by block until it comes to an END command in the data, or runs out of data.

VI. Program Performance

The L10 NLS program takes about two CPU minutes to run on TOPS20 on a DEC KL10 to decode the average document in fine detail mode. In this mode, the picture is about 1726 by 2100 pels, and takes about 204 disk pages to store.

We have a program which displays bit maps on an HP graphics terminal and have been able to display portions of documents. (not all of an 8.5" by 11" document will fit in the display). We can use the terminal's zoom capability to look at very small portions of the document.

References

[1] Weber, D. R., "An Adaptive Run Length Encoding Algorithm", International Conference on Communications, ICC-75, IEEE, San Francisco, California, June 1975.

[2] Mills, D. L., "Rapicom 450 Facsimile Data Decoding", WP2097/MD33E, COMSAT Laboratories, Washington D.C., undated.

[3] Casner, S. L., "Faxie", ISI Internal Memo, USC/Information Sciences Institute, February 1980.

[4] Postel, Jon, "Rapicom 450 Facsimile File Format", RFC 769, USC/Information Sciences Institute, September 1980.

Appendix

In this appendix is given the first portion of the data which comes from the fax machine, this same data in RFC 769 format, and some of this data decoded into a bitmap. The data is represented in octal octets.

The following is data of the form which comes out of the fax machine with length and command octets added:

```
114  70 142 171 330  13 377 377 377 371  53 200   0   5 125 125
125 125 125 125 125 125 125 125 125 125 125 125 125 125 125 125
125 125 125 125 125 125 125 125 125 125 125 125 125 125 125 125
125 125 125 125 125 125 125 125 125 125 125 125 125 125 125 125
125 125 125 125 125 125 125 125 125 121  21 261 114  71 142 171
330  40   0 102 326 270 152  42  42  44 111   0  42 151 267 122
366 110 237 102 211 365 111 171 336  51 244 247 377 377 111 362
177 377 377 377 377 377 377 377 377 376 104 213 241  41 111 377
111 337 377 377 377 377 377 377 377 377 377 377 377 163 301 361
377 377 377 377 360 177  12   0 114  71 142 171 330 141 137 177
377 344  10   0 160  23 301 160 137 376 204 352 135  27 353 264
  0  70 100   7  20  75   0   0   0   0   0 344   0   0   0   0
  0   0   0   0  34 275   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   7  41 310  34 200   0   0 344   0   0   0  71
 13 331 204   0 114  71 142 171 330 241 137  26 302 160   0  16
100  71   0 370 270 271   0 162   0  71 174 134 100 162   0  34
234 200 344   7 156 100   1 310  16 107  43 323 263 220 365 313
327  57 377 325 331  36  56  47 325 324 344   3 227  40  71  35
200   1 310   1 313 220   0   0   7 241 330   0   0 137 342 200
114  71 142 171 330 340  77  40 142 160   0   0   0   0 162  71
 73 162 376 276 234 277 376  67 265 301  16  20 171   1 311 313
346 377 321  75 256 113 245 377 262 160 136 247  13 251 350 374
270 236 235 217 136 203 220  75 166 166 364 177 305 366  72 107
 63 330 352 345 313 320  71  34 270  46  57   0
```

The following is the same data after put into RFC 769 format (with each data octet reversed and complemented):

```
114  70 271 141 344  57   0   0   0 140  53 376 377 137 125 125
125 125 125 125 125 125 125 125 125 125 125 125 125 125 125 125
125 125 125 125 125 125 125 125 125 125 125 125 125 125 125 125
125 125 125 125 125 125 125 125 125 125 125 125 125 125 125 125
125 125 125 125 125 125 125 125 125 165 167 162 114  71 271 141
344 373 377 275 224 342 251 273 273 333 155 377 273 151  22 265
220 355   6 275 156 120 155 141 204 153 332  32   0   0 155 260
  1   0   0   0   0   0   0   0   0 200 335  56 172 173 155   0
```

```
155   4   0   0   0   0   0   0   0   0   0   0   0  61 174 160
  0   0   0   0 360   1 257 377 114  71 271 141 344 171   5   1
  0 330 357 377 361  67 174 361   5 200 336 250 105  27  50 322
377 343 375  37 367 103 377 377 377 377 377 330 377 377 377 377
377 377 377 377 307 102 377 377 377 377 377 377 377 377 377 377
377 377 377 377  37 173 354 307 376 377 377 330 377 377 377 143
 57 144 336 377 114  71 271 141 344 172   5 227 274 361 377 217
375 143 377 340 342 142 377 261 377 143 301 305 375 261 377 307
306 376 330  37 211 375 177 354 217  35  73  64  62 366 120  54
 24  13   0 124 144 207 213  33 124 324 330  77  26 373 143 107
376 177 354 177  54 366 377 377  37 172 344 377 377   5 270 376
114  71 271 141 344 370   3 373 271 361 377 377 377 377 261 143
 43 261 200 202 306   2 200  23 122 174 217 367 141 177 154  54
230   0 164 103 212  55 132   0 262 361 205  32  57 152 350 300
342 206 106  16 205  76 366 103 221 221 320   1 134 220 243  35
 63 344 250 130  54 364 143 307 342 233  13 377
```

The following is the first part of the expanded bitmap of this data (there are about 4 scan lines here, or 2 pairs of scan lines):

```
177 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 367 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
337 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
377 377 377 377 377 377 377 374   0   4 327 377 377 377 377 377
374 377 356 377 177   0  10   0 201 200   0   0   0   0 100   0
  0   0   0   0   0   0   1 140   0   0   0   0   0   0   0   0
  0   0   0   0   0   0 204  10   0   0  10   0   0   0 100   0
 20  10   7 250   2   0  57 100 100   2 100 100 164   0  20  21
 31 310 153 137 377 377 377 377 177  32 176 344   2 200 216   0
  4   0 240   0   0  14  70   0   0   0   0   0   2  47 137 336
137 377 377 377 377 375 377 372  20 140  45 376 377 377 377 237
377 276 357 377 377 377 227 345 314 175  63 215 202   6 347 143
377 337 376  70 371 370 352 300 213 373 371 377 377 343  73 334
  0 207 315   3  33 111 377 167 337 377   1 323 365 177 377 177
377 374 377 135 377 377 365  67 343  55 377 377 377 377 357 377
377 377 377 377 377 377 203 377 236 175 376 236 337 273 347 377
```

```
376  77 377 377 377 377 377 377 377 377 377 377 300   0   0   0
200 102 177 377 277 377 377 377 376 377 366 365 173 302  12   0
 40 200   0   0   0   4 100   0   0   0   0   0   0   2   5 354
  0   0   0   0   0   0   0   0   4   0  10   0   0   0 200  10
 40  20   1   0 100   0 140   0  20 210 101 374   3 200 155 304
  0   6 100 103 376   0 120 121  31 332 243 177 377 377 377 377
377 233 377 354   0 241 217   1  30   0 240   0   0  12 150 202
 40   0   0   0  62  47 157 376 173 373 377 377 377 377 377 377
 20 141 321 376 377 377 377 327 377 376 377 377 377 377 237 216
316 375 167 215 202   6 300 143 377 237 374  70 175 330 377 304
255 373 153 377 377 353 377 104   0 267 315 203  13 311 177 377
377 377   1 223 367 377 373 167 377 376  77 137 377 345 165  67
 43  51 277 377 277 377 357 377 377 377 373 177 377 377 223 377
366 175 376 234 377 271 347 377 376 157 377 377 377 377 377 377
377 377 377 377 340   0   0   0   0   0 177 377  37 377 377 377
377 376 367 357 272 300   2   0   4   0   0   0   0   0   0   0
  0   0   0   0  20   0   1 144   0   0   0   0   0   0   4   4
  0   0 100   2 100  10 201  10   0  20  75   0   0  40 142   0
  0  74 341 234 103   4 157 300   0   2   0 141 372   0   0  20
 30 376  55 277 177 377 377 367 377 371 376 100  15  61  16 200
 30   0  40   0   0   0 311 200  24   0   0   0  62  55 377 316
367 347 377 357 377 377 377 377 170 305   5 276 377 377 377 357
377 377 377 377 377 177 377 377 357 177 377  76 207 246 340 147
376 336 356  10  17 320 105 235 275 377 377 373 377 347 335 317
 50  77 377 353  75 333 377 377 377 377 363 337 343 277 356 171
  7 357  76 216 377 211 207 176 257 217 377 377 367 357 357 277
377 357 377 377 377 375 367 377 377 377 377 375 377 377 356 377
366 377 377 377 377 377 377 377 377 377 377 377 340   0   0   0
  0  44 373 377  77 377 377 177 177 377 377 337 376 170 173   0
  0   0 100   0   1  10   0   0   0   0   0 200 160   0 223 160
300   0   0   0   0   0   0   6 100 220   0   0 140   4   3  30
121  20 351 300 206  74 167   0  30  64  41 234 172  30 175 300
  4  32   4 345 367 200 103  60 177 372 177 233 377 377 377 377
376 125 207 210 233  21 364 361 277   1  50  16 140 120  41 335
377 306 214  10  67 377 373 377 377 377 377 377 377 367 377 377
377 363 277 377 377 377 377 377 267 177 377 377 377 377 377 237
377 377 377  77 377 377 355 373
```